NEZHA: Efficient Domain-Independent Differential Testing

Theofilos Petsios*, Adrian Tang*, Salvatore Stolfo, Angelos D. Keromytis, and Suman Jana

IEEE Security & Privacy (Oakland) 2017
*Joint primary authors

Columbia University
Differential Testing
Differential Testing

- Fuzzing: memory corruption bugs
- Differential testing: logic bugs
Differential Testing

- Multiple apps of the same functionality
- Applications usually follow some specification/standard
Differential Testing

- Multiple apps of the same functionality
- All usually to follow some specification/standard
- Deviations from the specifications/standards likely to be bugs
Differential Testing

- Multiple apps of the same functionality
- All usually to follow some specification/standard
- Deviations from the specifications/standards likely to be bugs
Differential Testing

- Multiple apps of the same functionality
- All usually to follow some specification/standard
- Deviations from the specifications/standards likely to be bugs
- Applicable in different domains (e.g., compiler testing)
Key challenges

• Existing tools are domain-specific

• Inefficient input generation
Goal of NEZHA

Efficient domain-independent differential testing
Domain-Independent Evolutionary Testing

Seed Inputs

Input Generation
Guidance

New Inputs

Application
Domain-Independent Evolutionary Testing

Seed Inputs

Input Generation Guidance

New Inputs

Runtime Monitoring

Application
Domain-Independent Evolutionary Testing

Seed Inputs

Input Generation Guidance

State Information

New Inputs

Runtime Monitoring

Application
Evolve an input corpus that is guided based on an analysis engine
Evolve an input corpus that is \textit{guided} based on an \textit{analysis engine}
Code Coverage - Single-App

All possible code paths
Code Coverage - Single-App

Input Corpus

Per-Input Coverage
Code Coverage - Single-App

Input Corpus

Per-Input Coverage

- All possible code paths
- Code coverage - Global
- Code coverage - Input

Input 1
Code Coverage - Single-App

Input Corpus

Per-Input Coverage

Input 1

All possible code paths

Code coverage - Global

Code coverage - Input
Code Coverage - Single-App

Input Corpus

Per-Input Coverage

All possible code paths

Code coverage - Global

Code coverage - Input
Code Coverage - Single-App

Input Corpus

Input 1

Per-Input Coverage

All possible code paths

Code coverage - Global

Code coverage - Input
Input Corpus

Input 1

Per-Input Coverage

All possible code paths

Code coverage - Global

Code coverage - Input
Code Coverage - Single-App

All possible code paths

Input 2

Code coverage - Global

Code coverage - Input
Code Coverage - Single-App

All possible code paths

Input Corpus

Input 1

Input 2

Per-Input Coverage

Code coverage - Global

Code coverage - Input
Code Coverage - Single-App

Input Corpus

<table>
<thead>
<tr>
<th>Input 1</th>
<th>Input 2</th>
</tr>
</thead>
</table>

Per-Input Coverage

- All possible code paths
- Code coverage - Global
- Code coverage - Input

Input 3
Code Coverage - Single-App

Input Corpus

Input 1
Per-Input Coverage

Input 2

All possible code paths

Code coverage - Global

Code coverage - Input
Code Coverage - Single-App

Input Corpus

- Input 1
- Input 2
- Input 3

Per-Input Coverage

- Code coverage - Global
- Code coverage - Input

All possible code paths

Input 3
# Code Coverage - Single-App

<table>
<thead>
<tr>
<th>Input Corpus</th>
<th>Input 1</th>
<th>Input 2</th>
<th>Input 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Per-Input Coverage</td>
<td><img src="coverage1" alt="coverage" /></td>
<td><img src="coverage2" alt="coverage" /></td>
<td><img src="coverage3" alt="coverage" /></td>
</tr>
</tbody>
</table>

- **All possible code paths**
- Yellow: Code coverage - Global
- Gray: Code coverage - Input

Input 4
Code Coverage - Single-App

- All possible code paths
- Code coverage - Global
- Code coverage - Input

Input Corpus | Input 1 | Input 2 | Input 3
---|---|---|---
Per-Input Coverage | | | |
Domain-Independent Evolutionary Testing

Seed Inputs → State Information → Input Generation Guidance → New Inputs → Application → Runtime Monitoring
Evolutionary Differential Testing - Multiple-Apps

What are the options to driving input generation?

1. Use program states solely from single application, like most modern fuzzers
2. Use global program states combined across all applications
3. Re-design guidance engine geared towards differential testing
Evolutionary Differential Testing - Multiple-Apps

What are the options for driving input generation?

1. Use program states solely from \textit{single application}, like most modern fuzzers

2. Use \textit{global} program states \textit{combined} across all applications

3. Re-design guidance engine \textit{geared towards differential testing}
Key Insight

Techniques that work well in the context of single application testing may not be optimal for differential testing!
Multi-App Code Coverage

App1

App2

Input Corpus

Per-Input Coverage

All possible code paths

Code coverage - Global

Code coverage - Input
Multi-App Code Coverage

App1

App2

Input Corpus

Per-Input Coverage

Input 1

All possible code paths

Code coverage - Global

Code coverage - Input
Multi-App Code Coverage

Input Corpus

Per-Input Coverage

App1

App2

Input 1

All possible code paths

Code coverage - Global

Code coverage - Input
Multi-App Code Coverage

Input Corpus

Per-Input Coverage

All possible code paths

Code coverage - Global

Code coverage - Input
Multi-App Code Coverage

App1

App2

Input Corpus

Input 1

Per-Input Coverage

Input 1

All possible code paths

Code coverage - Global

Code coverage - Input
Multi-App Code Coverage

- App1
- App2

Input Corpus

Input 1

Per-Input Coverage

Input 2

- All possible code paths
- Code coverage - Global
- Code coverage - Input
Multi-App Code Coverage

Input Corpus: All possible code paths
Per-Input Coverage:
- Code coverage - Global
- Code coverage - Input
Multi-App Code Coverage

Input Corpus

Per-Input Coverage

Input 1

Input 2

App1

App2

Input 3

All possible code paths

Code coverage - Global

Code coverage - Input
Multi-App Code Coverage

Input Corpus

Per-Input Coverage

Input 1

Input 2

All possible code paths

Code coverage - Global

Code coverage - Input

Discard!
Multi-App Code Coverage

Input 4

All possible code paths

Code coverage - Global

Code coverage - Input
Multi-App Code Coverage

App1               App2

Input Corpus  Input 1  Input 2

Per-Input Coverage

All possible code paths

Code coverage - Global

Code coverage - Input

Discard!
Multi-App Code Coverage

Input 3

App1

Input 3

App2
Multi-App Code Coverage

Input 3

Input 4

App1

App2
Multi-App Code Coverage

- These inputs exercise disproportionate code regions in the two apps
- This disproportion might imply differences in handling logic
- Retaining them in corpus speed up process of finding discrepancies
Relative program behavior is important in this context!
δ-diversity: a new approach to guided differential testing
Differential Testing under $\delta$-diversity

- Obtain State Information
  - White-box (e.g., at compile time)
  - Gray-box (e.g., using Dynamic Binary Instrumentation)
  - Black-box (e.g., only examining system response to inputs)

- Behavioral Diversity
Differential Testing under δ-diversity
Differential Testing under $\delta$-diversity
Differential Testing under $\delta$-diversity
Differential Testing under $\delta$-diversity

Behavioral Asymmetries

OpenSSL
LibreSSL
wolfSSL
Gnu TLS
Differential Testing under δ-diversity

Behavioral Asymmetries
Differential Testing under $\delta$-diversity

- Two examples:
  - Gray-box
  - Black-box

- Both outperform code coverage
Path $\delta$-diversity: gray-box

Keep track of **unique edges**
Path $\delta$-diversity: gray-box

Input 1 ✓

Input 2 ✓

Input 3 ✗

Keep track of unique edges
Output δ-diversity: black-box

<table>
<thead>
<tr>
<th>Input</th>
<th>App1</th>
<th>App2</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>0x0</td>
<td>0x0</td>
</tr>
<tr>
<td>✓</td>
<td>0x1</td>
<td>0xdead</td>
</tr>
<tr>
<td>✗</td>
<td>0x0</td>
<td>0x0</td>
</tr>
<tr>
<td>✓</td>
<td>0x0</td>
<td>0x0</td>
</tr>
<tr>
<td>✓</td>
<td>0x0</td>
<td>0x0</td>
</tr>
<tr>
<td>✗</td>
<td>0xbabe</td>
<td>0x0</td>
</tr>
<tr>
<td>✓</td>
<td>0x0</td>
<td>0x0</td>
</tr>
</tbody>
</table>

All possible code paths
Return values
Error codes
Exception messages
δ-diversity

- Domain Independence
- Efficient differential guidance
Implementation

- NEZHA prototype
- Gray-box and black-box $\delta$-diversity metrics
  - Path $\delta$-diversity (fine & coarse)
  - Output $\delta$-diversity
- Domain-independent input generation
  - Evolutionary, feedback-guided
- Built upon libFuzzer with NEZHA-specific hooks
- 1545 lines of C++
Use cases

- SSL libraries
- PDF readers
- ClamAV & XZ Parsers
Use cases

- SSL libraries
  - LibreSSL
  - OpenSSL
  - wolfSSL
  - GnuTLS

- PDF readers
  - pdf
  - e

- ClamAV & XZ Parsers
Certificate Verification Discrepancies

One library accepts one certificate, while another rejects it with an error code.

<table>
<thead>
<tr>
<th></th>
<th>LibreSSL</th>
<th>BoringSSL</th>
<th>wolfSSL</th>
<th>mbedTLS</th>
<th>GnuTLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenSSL</td>
<td>10</td>
<td>1</td>
<td>8</td>
<td>33</td>
<td>25</td>
</tr>
<tr>
<td>LibreSSL</td>
<td>-</td>
<td>11</td>
<td>8</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>BoringSSL</td>
<td>-</td>
<td>-</td>
<td>8</td>
<td>33</td>
<td>25</td>
</tr>
<tr>
<td>wolfSSL</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>mbedTLS</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>31</td>
</tr>
</tbody>
</table>

Unique pair-wise discrepancies (based on error code tuples)
NEZHA vs domain-specific frameworks

- 52x more discrepancies than Frankencerts
- 27x more discrepancies than Mucerts
NEZHA vs popular evolutionary fuzzers

- Adapted popular evolutionary fuzzers for differential testing
  - Code coverage in single application
  - Global code coverage
- 6x more discrepancies than testing on a single application
- 30% more discrepancies than modified libFuzzer

![Graph showing comparison between NEZHA and popular evolutionary fuzzers]
Sample Bugs uncovered by NEZHA
*(disclosed and patched)*
# Experimental Setting

<table>
<thead>
<tr>
<th>Application Category</th>
<th>Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL Libraries</td>
<td>OpenSSL, LibreSSL, BoringSSL, GnuTLS, wolfSSL, mbedTLS</td>
</tr>
<tr>
<td>PDF Readers</td>
<td>Evince PDF, MuPDF, Xpdf</td>
</tr>
<tr>
<td>Parsers</td>
<td>ClamAV vs binutils, ClamAV vs xz</td>
</tr>
</tbody>
</table>
Bug 1: Malicious ELF can evade ClamAV detection

ClamAV (ELF parsing engine)

```c
static int cli_elf_fileheader(...) {
    ...
    switch(file_hdr->hdr64.e_ident[4]) {
    case 1:
        ...
    case 2:
        ...
    default:
        ...
        return CL_EFORMAT;
    ...
```
Bug 1: Malicious ELF can evade ClamAV detection

ClamAV (ELF parsing engine)
**Bug 1: Malicious ELF can evade ClamAV detection**

**CLAMAV (ELF parsing engine)**

```c
static int cli_elf_fileheader(...) {
    switch(file_hdr->hdr64.e_ident[4]) {
        case 1:
            ...
        case 2:
            ...
        default:
            ...
            return CL_EFORMAT;
    }
    ...
}
```

**LINUX ELF loader**

```c
static int load_elf_binary(struct linux_binprm *bprm) {
    ...
    retval = -ENOEXEC;
    if (memcmp(loc->elf_ex.e_ident, ELFMAG, SELF MAG) != 0)
        goto out;
    if (loc->elf_ex.e_type != ET_EXEC &&
        loc->elf_ex.e_type != ET_DYN)
        goto out;
    if (!elf_check_arch(&loc->elf_ex))
        goto out;
    ...
}
```
Bug 1: Malicious ELF can evade ClamAV detection

ClamAV (ELF parsing engine)

Linux ELF loader
Bug 2: LibreSSL misinterprets time in ASN.1 format

Time fields can be formatted in 2 ways:

**UTC**: YYMMDDHHMMSSSZ (13 char long)

**GMT**: YYYYMMDDHHMMSSSZ (15 char long)
Bug 2: LibreSSL misinterprets time in ASN.1 format

Time fields can be formatted in 2 ways:

**UTC**: YYMMDDHHMMSSSZ (13 char long)

**GMT**: YYYYMMDDHHMMSSSZ (15 char long)

LibreSSL ignores the ASN.1 time format tag, and determines format based on length of field
Bug 2: LibreSSL misinterprets time in ASN.1 format

LibreSSL ignores the ASN.1 time format tag, and determines format based on length of field
Bug 2: LibreSSL misinterprets time in ASN.1 format

LibreSSL ignores the ASN.1 time format tag, and determines format based on length of field
Bug 2: LibreSSL misinterprets time in ASN.1 format

LibreSSL ignores the ASN.1 time format tag, and determines format based on length of field

Jan 1 01:01:00 2012 GMT can interpreted as Dec 1 01:01:01 2020 GMT
Conclusions

- δ-diversity outperforms code coverage for differential testing

- NEZHA: Domain independent, efficient differential testing

- Differential testing should be integrated, when possible, into the testing cycle

https://github.com/nezha-dt
Backup Slides
NEZHA: Architecture

Instrumentation Module

Programs (Original)

Input Corpora

Initial Seeds

NEZHA Engine

- Differential Execution
- Input Mutation
- Corpus Refinement
- Guidance Engines

NEZHA Runtime Library

- Dynamic Coverage Information
- Program Return Values

Discrepancy Logging

Application Address Space
#include <openssl/evp.h>

extern "C"
int LLVMFuzzerTestOneInput(const uint8_t *buf, size_t len) {
    const uint8_t *bufp = buf;
    EVP_PKEY_free(d2i_AutoPrivateKey(NULL, &bufp, len));
    return 0;
}
Nezha: Architecture

The diagram illustrates the architecture of Nezha, a fuzz testing tool. It includes the following components:

- Engine
  - UpdateDiff
  - RunOne
  - LLVMTestOneInput
  - LLVMFuzzerCovBuffers
  - LLVMFuzzerRetVals

- Tested Applications
  - Fuzz_TestStart
  - Process_i (Data)
  - Fuzz_TestEnd

- Input corpus

The processes are labeled as follows:

1. Process_i (Data)
2. Fuzz_TestStart
3. Process_i (Data)
4. Fuzz_TestEnd
5. Process_i (Data)
6. Fuzz_TestEnd
7. Engine
8. Input corpus
Discrepancy Distribution for SSL/TLS Libs

Same Inputs / Different mode SSL libraries tested

- Output $\delta$-diversity
- Path $\delta$-diversity
- Global Coverage
Number of unique discrepancies

- Global coverage (modified libFuzzer)
- Path $\delta$-diversity (coarse)
- Path $\delta$-diversity (fine)
- Output $\delta$-diversity